UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Gestão de Acesso a Dados no Back-End: Controlo de Acesso Baseado em Funções e Geração Automática de Relatórios

Rui Filipe Vilaça Oliveira Peixoto

Mestrado em Engenharia Informática
Especialização em Engenharia de Software

Versão Pública

Trabalho de Projeto orientado por:
Prof. Doutora Andreia Filipa Torcato Mordido

2020

Agradecimentos

Primeiramente, gostaria de agradecer a toda a minha família, em particular: à minha namorada, que me apoia incansável e incondicionalmente; aos meus pais, que, apesar das adversidades da vida, sempre me encorajaram e ajudaram a seguir os meus sonhos; aos meus irmãos pelo companheirismo e boa disposição em todos os momentos; à minha madrinha por estar sempre presente e disponível para tudo.

Um especial agradecimento também aos pais, avós e restante família da minha namorada, que desde o primeiro dia me acolheram e apoiaram em todos os níveis.

Agradeço à Faculdade de Ciências da Universidade de Lisboa e a todos os meus professores, pelo rigor de ensino e todos os anos de constante aprendizagem ao longo deste percurso académico. Um obrigado especial à minha orientadora, que se mostrou sempre interessada e solícita, desempenhando um papel deveras importante no sucesso deste projeto.

Gostaria ainda de agradecer a todos os meus amigos mais próximos, incluindo o grupo de amigos do Secundário, que mesmo longe se mantém sólido e unido, e aos meus colegas de licenciatura e mestrado, nomeadamente André, Paulo, Inês e Micael, pelo apoio e todos os trabalhos que desenvolvemos juntos.

Por último, mas não menos relevante, gostaria de agradecer à empresa que me recebeu para a realização deste projeto e que muito contribuiu para meu crescimento profissional. Aos meus colegas de equipa, em especial ao Simão, também aluno do mestrado na FCUL, com o qual partilhei bons momentos de entre-ajuda. Ao Marco, líder técnico do projeto, pela sua constante disponibilidade e por ser um mentor dedicado que tanto me ensinou. À Ana, a minha orientadora na empresa, pela flexibilidade e confiança que depositou em mim e no meu trabalho.

A todos, o meu muito obrigado!

À Família

Resumo

Atualmente, é guardada uma quantidade significativa de informação em aplicações disponibilizadas *online*, pelo que se tornou imperativo proteger os dados com um elevado grau de confiança, para que não possam ser furtados ou acedidos por entidades que não tenham autorizações para tal.

Esta dimensão de segurança é a principal vertente do presente projeto, realizado no âmbito do mestrado em Engenharia Informática, que consiste no desenho e criação de vários serviços para um software de certificação ambiental, designado por FSIM.

Dos desenvolvimentos realizados para o efeito, descritos ao longo deste trabalho, realçam-se dois: um sistema de controlo de acessos e um servidor de relatórios. O sistema de controlo de acessos, baseado em funções, permite intermediar a gestão de permissões dos utilizadores. Apesar de ser aplicado ao FSIM, este modelo pode ser empregue em inúmeras outras aplicações Java Spring. O servidor de relatórios Jasper possibilita a geração automática de relatórios, atendendo à regularidade e à necessidade de utilização de um serviço desta natureza por parte de empresas em auditorias. Este servidor impôs-se para substituir o Jasper Server, devido aos constrangimentos deste relativos ao controlo de versões do método de integração contínua adotado pela equipa.

Neste trabalho, são igualmente expostas as diferentes abordagens de gestão de utilizadores dentro de uma aplicação, entre as quais se destaca a atribuição de perfis aos utilizadores com determinadas funções, associadas à realização de ações específicas.

Para uma correta integração contínua, foram desenvolvidos testes unitários, que permitem garantir as funcionalidades implementadas no ciclo de vida do software.

Com base nos devidos requisitos de segurança, a robustez e a fiabilidade foram dos principais aspetos a ter em conta, pelo que se realizaram e analisaram ainda vários testes de penetração.

Palavras-chave: Controlo de Acesso Baseado em Funções, Testes de Penetração, Spring, Jasper, Gestão de Utilizadores

Abstract

Nowadays, a significant amount of information is stored in applications available online, thus it has become imperative to secure data with a high degree of reliability, so that it cannot be stolen or accessed by entities without proper authorization.

This dimension of security is the main focus of this project, set forth within the scope of the master's degree in Informatics Engineering, which consists of the design and deployment of several services for a software of environmental certification, called FSIM.

In this document, we present the developments for this software in detail, from which we highlight two: an access control system and a report server. The access control system, developed over a role-based model of access control, allows intermediation of user permissions management. Despite being applied to FSIM, this model can be used in other Java Spring applications. The Jasper report server enables the automatic generation of reports, meeting the regularity and the need of companies to use a service of this nature in audits. The Jasper Server needed to be replaced by this server, due to its constraints related to version control of the continuous integration method adopted by the team.

The different user management approaches within an application are also studied in this work, among which the attribution of profiles to users with certain roles in order to execute specific actions stands out.

Furthermore, for a correct continuous integration, unit tests were developed, aimed to guarantee the functionalities implemented in the software's life cycle.

Based on the appropriate safety requirements, robustness and dependability were among the main aspects to be taken into account. Hence, several penetration tests were also performed and analysed.

Keywords: Role Based Access Control, Pen-Testing, Spring, Jasper, User Management

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

BA Back-Office Administrativo.

BE Back-End.

CRUD Create Read Update Delete.

DMS Document Management System.

DTO Data Transfer Objects.

FE Front-End.

JSON JavaScript Object Notation.

MVC Model View Controller.

PGOA Plano de Gestão Operacional Ambiental.

REST Representational State Transfer.

SOAP Simple Object Access Protocol.

XML Extensible Markup Language.

YAML YAML Ain't Markup Language.

Capítulo 1

Introdução

Este trabalho foi realizado no âmbito do mestrado em Engenharia Informática (especialização em Engenharia de Software) e tem como objeto o desenvolvimento de um projeto que surgiu do protocolo acordado entre a Faculdade de Ciências da Universidade de Lisboa e a empresa TrustSystems.

A proposta de trabalho centrou-se numa aplicação, neste trabalho denominada de FSIM, destinada a facilitar o processo de certificação ambiental, de acordo com as normas internacionais.

O software, cuja apresentação formal foi no passado dia 25 de Junho de 2020, consiste num necessário instrumento de gestão de informação, nomeadamente documental e cartográfica, para os grupos de certificação. Está atualmente ao serviço de três empresas, no entanto espera-se que a plataforma seja utilizada por mais organizações do mesmo domínio.

O FSIM não só tem um grande valor económico para os clientes, como contribui para tornar o setor de exploração de recursos naturais mais sustentável. Crê-se que a certificação ambiental produz impactos positivos no meio ambiente, em comparação com procedimentos e explorações não certificadas e realizadas de forma convencional [46], desempenhando um papel importante na preservação da riqueza da fauna, na conservação da vasta diversidade da flora e na melhoria dos ecossistemas, nomeadamente no que concerne a redução da poluição atmosférica, diminuição da desflorestação nas operações florestais, entre outros [12].

1.1 Motivação

Atualmente, a informação sensível é tendencialmente guardada em aplicações informáticas, que constituem um alvo fácil para entidades com intuitos maliciosos. Existem inúmeros ataques informáticos capazes de furtar massivas quantidades de informação. A título de exemplo, este ano, a companhia aérea *Easy-Jet* sofreu um ataque informático sofisticado que roubou dados de 9 milhões de utilizadores, incluindo endereços de email e até alguns cartões de crédito [9].

A cibersegurança é também um problema em Portugal, tendo-se intensificado os ataques durante o cenário pandémico de Covid-19. O Centro Nacional de Cibersegurança divulgou um aumento de 176% em Março de 2020, relativamente ao período homólogo do ano anterior [10]. Neste sentido, é imperativo tomar medidas de segurança para que todos os dados se mantenham protegidos e inacessíveis a quem não tem permissões.

Disponibilizar plataformas online constitui naturalmente um desafio, pelo facto de se encontrarem na Internet com acesso público e terem requisitos complexos para o seu funcionamento.

Torna-se, assim, necessário criar sistemas e mecanismos capazes de manter a integridade e segurança da informação, para que não possa ser acedida ou furtada por entidades maliciosas.

1.2 Objetivos

Os principais objetivos traçados para o projeto, analisados pormenorizadamente no presente relatório, passaram por: tornar a aplicação FSIM mais robusta e segura de forma a minimizar o risco de roubo de informação, permitir a geração automática de documentos com alto valor para os clientes e criar alguns serviços na API para alimentar as aplicações Front-End.

Com vista a tornar a aplicação mais segura, seria necessário o desenvolvimento de parte integrante do módulo de gestão de utilizadores, mais concretamente na gestão de permissões, e, também de elevada importância, a realização testes de penetração para aferir a segurança da aplicação e perceber se haveria melhorias a serem implementadas.

Para a geração automática de documentos, tendo em conta o fim a que se destina a aplicação, impunha-se a geração de relatórios automáticos. A inclusão desta funcionalidade no processo de integração contínua, requeria a criação de um servidor de relatórios

Jasper para o efeito.

Com o intuito de complementar a aplicação, almejava-se a criação de vários serviços na API, cujos desenvolvimentos poderiam desempenhar um papel fundamental no aumento do meu conhecimento sobre a plataforma, sobretudo a nível arquitetural, e sobre os padrões de código adotados pela equipa.

1.3 Contribuições

Os meus desenvolvimentos foram maioritariamente focados na gestão de privilégios dos utilizadores e na criação de um servidor de relatórios *Jasper*. No entanto, durante o projeto, foram-me atribuídas outras tarefas e responsabilidades (abordadas adiante). Deste modo, as minhas contribuições principais para o desenvolvimento da aplicação foram as seguintes:

- Desenvolvimento e melhoria de serviços de um módulo abstrato à aplicação, capaz de tratar da gestão de utilizadores e dos seus privilégios;
- Construção de um serviço de importação automática de dados pré-existentes para facilitar a integração desta plataforma com o sistema do cliente;
- Desenvolvimento de um serviço agregador que colecciona toda a informação necessária para a geração automática de um documento utilizado na certificação;
- Elaboração de uma plataforma para criar relatórios *Jasper* (relatórios de membros, relatórios de erros, entre outros), que possa ser incluído no *Git* para proporcionar controlo de versões;
- Criação de um serviço que lê as configurações de temas de projetos QGIS e cria um ficheiro JSON utilizado para uma aplicação denominada QWC2;
- Testes unitários da aplicação, para garantir a correção e robustez do código;
- Testes às vulnerabilidades do módulo através de *pen testing*, para verificar se a informação poderia ou não ser acedida por outras entidades.

Em suma, as minhas contribuições estão presentes por vários módulos e serviços do Back-End da aplicação, tendo maior incidência no módulo de gestão de utilizadores e no servidor de relatórios.

1.4 Instituição de Acolhimento e Equipa de Desenvolvimento

Este projeto foi desenvolvido na *TrustSystems*, uma consultora que se insere na área da Segurança de Informação. É especializada no fornecimento de soluções, em ambientes *corporate* e governamentais, de modo a garantir a segurança dos ativos de informação. A sua atuação abrange quatro áreas: proteção de informação, serviços de segurança, soluções de parceiros e I&D [40].

A empresa pertence ao grupo Inoweiser, cujo principal propósito consiste em usar a tecnologia e a inovação para satisfazer as necessidades dos seus clientes. O grupo tem mais de quinze anos de experiência em mercados nacionais e internacionais e está presente em mais de 16 países [15].

A equipa de desenvolvimento, na qual estou inserido para a realização deste projeto, é composta por mais dez elementos: o líder técnico Marco Morado, um programador Back-End sénior, dois programadores Back-End juniores, dois programadores Front-End, dois testers, um analista funcional e um programador *Jasper-Reports*. Para além da equipa de desenvolvimento, destaca-se também a Gestora de Projeto Ana Guimarães.

1.5 Estrutura do Documento

O relatório está organizado da seguinte forma:

- **Capítulo 1 – Introdução:** Apresentação do projeto, motivação, objetivos, contribuições e instituição de acolhimento.
- **Capítulo 2 – Trabalho Relacionado:** Tecnologias, *frameworks* e outras abordagens utilizadas para a resolução do problema.
- **Capítulo 3 – Análise:** Análise da arquitetura existente e dos módulos/serviços necessários no âmbito do projeto.
- **Capítulo 4 – Desenho:** Desenho detalhado do módulo de gestão de permissões dos utilizadores e do servidor de relatórios.
- **Capítulo 5 – Implementação:** Implementação da arquitetura da *cloud* e outros desenvolvimentos propostos.

- **Capítulo 6 – Testes e Resultados:** Resultados dos testes de penetração, testes funcionais e testes unitários da aplicação.
- **Capítulo 7 – Conclusão:** Conclusões do projeto e trabalho futuro.

Capítulo 2

Trabalho Relacionado

O presente capítulo descreve metodologias existentes para a gestão de utilizadores, bem como tecnologias relevantes para a realização deste projeto, centrado no desenvolvimento de um serviço de gestão de permissões, de um servidor de relatórios Jasper e de testes de penetração.

Serão abordadas matérias relativas à *framework* Spring e a alguns dos seus módulos, ao JasperReports, aos Bearer Tokens como meios de gestão de sessões HTTP e ainda aos softwares atuais para a realização dos testes de penetração.

2.1 Gestão de Utilizadores e Permissões

Existem duas tarefas essenciais na gestão de utilizadores: a autenticação e a autorização. Apesar de serem conceitos que por vezes se confundem, a distinção é simples – a autenticação trata de identificar um utilizador, enquanto que a autorização verifica se um utilizador tem permissões para realizar uma ação [66].

Neste projeto, propusemo-nos a criar um mecanismo de gestão de permissões de utilizadores na aplicação. Na literatura é possível encontrar várias abordagens e metodologias para o efeito, como por exemplo Controlo de Acesso Baseado em Funções e Controlo de Acesso Baseado em Atributos.

O Controlo de Acesso é apenas uma das soluções de segurança informática mais destacada. Permite limitar o que um utilizador legítimo consegue fazer perante um sistema computacional [65], isto é, cada vez que um utilizador entra num sistema computacional, dá-se a ativação automática do Controlo de Acesso [49] e as ações do utilizador são restringidas, de forma a prevenir falhas de segurança.

Há diversas políticas de controlo de acesso que podem ser aplicadas, consoante diferentes critérios sobre o que deve ou não ser permitido e diferentes definições no que toca à garantia da segurança [64], entre as quais:

- Listas de Controlo de Acesso;
- Controlo de Acesso Baseado em Funções;
- Controlo de Acesso Baseado em Atributos;
- Controlo de Acesso Baseado em Latência;
- Controlo de Acesso Baseado em Organizações.

Esta secção serve o propósito de expor e comparar os métodos que também se adequariam à ferramenta FSIM, a salientar, Listas de Controlo de Acesso, Controlo de Acesso Baseado em Funções (*Role-Based Access Control*) e Controlo de Acesso Baseado em Atributos.

2.1.1 Listas de Controlo de Acesso

Listas de Controlo de Acesso constituem um modelo baseado em permissões. São usadas para descrever as ações autorizadas (permissões que representam diferentes tipos de acesso a um projeto) de um utilizador relativamente a recursos de um sistema [58].

Para uma melhor compreensão deste modelo, clarifico os seguintes conceitos:

- Utilizador – Pessoa que utiliza um sistema ou aplicação.
- Permissão – Autorização para realizar uma operação sobre um sistema ou aplicação.

Cada utilizador pode conter várias permissões sobre o sistema (figura 2.1). Esta relação pode ser feita utilizando um ficheiro de configuração ou uma estrutura de Base de Dados.

Figura 2.1: Modelo de Acesso Baseado em Permissões

Referenciando um caso divulgado no artigo *Implementing Role Based Security in a Web App* [66], uma equipa de programadores da empresa *Bluecore Engineering* começou por desenhar um sistema baseado em lista de acessos, em que para cada recurso de uma aplicação *web* configuraram o seguinte JSON:

```
{
    "ler":[
        "Diogo",
        "Pedro"
    ],
    "escrever":[
        "Joana"
    ]
}
```

Com estes moldes, dado que o Diogo e o Pedro estão na lista de leitura e a Joana está na lista da escrita, se a Joana tentar escrever no recurso e utilizar, por exemplo, o método HTTP Post, será bem sucedida e obterá um código HTTP 200. No entanto, se for o Pedro, resultará num código de erro que deverá ser 403 - Não autorizado.

A implementação deste modelo é simples, contudo, não é comportável se a aplicação tiver muitas regras de negócio, no que diz respeito às autorizações disponíveis, ou um grande número de utilizadores. Ao optar por este modelo, para cada novo utilizador do sistema é necessário verificar e atribuir todas as suas permissões.

2.1.2 Controlo de Acesso Baseado em Funções

O controlo de acesso baseado em funções é uma tecnologia que tem atraído muita atenção, particularmente em grandes aplicações comerciais, devido ao seu potencial para reduzir custos, uma vez que diminui a complexidade da gestão de privilégios [49], nomeadamente em relação à facilidade com que um utilizador pode mudar de funções num sistema, ficando automaticamente agregado a uma série de permissões caso mude de cargo numa empresa.

Para além dos conceitos abordados anteriormente (secção 2.1.1), este modelo introduz um novo termo denominado de função. Uma função *(role)* é um cargo atribuído a uma pessoa (utilizador), que lhe permite realizar determinadas ações sobre o sistema, e funciona como um agregador de permissões. Uma função pode ser, por exemplo, uma posição de trabalho numa empresa.

As permissões são associadas às funções e os utilizadores têm de ter funções atribuídas para adquirir permissões [61]. A figura 2.2 demonstra que os utilizadores estão ligados a funções e as funções, por sua vez, estão ligadas a permissões. Um utilizador pode conter

mais do que uma função e as funções deverão ser também um conjunto de permissões.

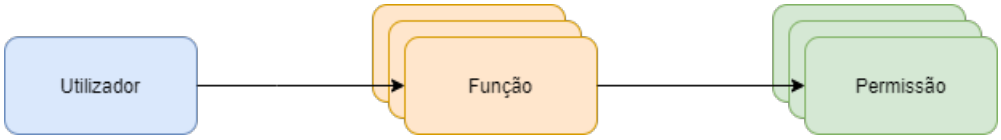

Figura 2.2: Modelo de Acesso Baseado em Funções

Este processo simplifica significativamente a gestão de permissões, dado que as funções podem ser criadas para os vários cargos de trabalho numa organização e atribuídas aos utilizadores com base nas suas responsabilidades e qualificações. As funções dos utilizadores são facilmente trocadas, se necessário, e podem receber novas permissões à medida que novas aplicações e sistemas são incorporados [61]. Cada vez que é criado um novo utilizador, este deverá receber funções consoante o seu cargo.

Para implementar este modelo, é possível utilizar uma Base de Dados ou um ficheiro de configuração.

No primeiro caso, é necessária a criação de 3 tabelas distintas: Utilizadores, Funções e Permissões. As associações seguem o diagrama UML presente na figura 2.3.

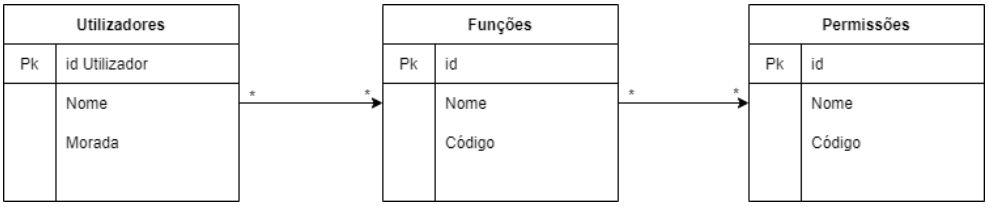

Figura 2.3: Diagrama UML de Acesso Baseado em Funções

A segunda opção passa pela criação de um ficheiro de configuração, por exemplo YAML ou JSON, que poderá seguir o seguinte formato dos blocos de código:

```
{
    "Administrador":[
        "Ana",
        "Pedro"
    ],
    "Cliente":[
        "Catarina"
    ]
}
```

No bloco JSON acima, existem duas funções - administrador e cliente - que são atribuidas aos utilizadores. A Ana e o Pedro possuem a função de administrador e a Catarina a função de cliente.

```
{
   "Administrador":[
      "Ler",
      "Escrever"
   ],
   "Cliente":[
      "Ler"
   ]
}
```

Neste bloco, as funções estão atribuídas a permissões. Assim, os administradores podem ler e escrever, enquanto que os clientes apenas podem ler.

Em suma, juntando ambos os blocos, é possível observar que a Ana e o Pedro podem ler e escrever e a Catarina apenas pode ler.

Existem ainda algumas regras fundamentais no desenho de um modelo baseado em funções [32]:

- **Atribuição de Funções:** um utilizador pode realizar uma ação somente se tiver uma função;
- **Autorização de Funções:** um utilizador apenas pode ter uma função se lhe for dada autorização para a mesma;
- **Autorização de Permissões:** um utilizador apenas pode obter uma permissão se constar numa das suas funções.

Como podemos inferir, este modelo é de implementação simples e corresponde, de forma correta, à complexidade exigida pelos requisitos de gestão de permissões das aplicações atuais. No entanto, o artigo *Implementing Role Based Security in a Web App* [66] refere que este processo exige que as equipas de Front-End tenham conhecimento das permissões necessárias para cada ação (dado que não podem apresentar botões ou páginas a um utilizador que não tenha permissão de visualização) e que os gestores da aplicação se debrucem sobre as permissões que cada função poderá ter na aplicação, para que no final as funções contenham as permissões corretas. Este pode ser um ponto negativo, uma vez que se essa gestão não for bem feita, quer pelos gestores quer pelo Front-End, a aplicação pode ficar vulnerável.

2.1.3 Controlo de Acesso Baseado em Atributos

Segundo os autores Junbeom Hur e Dong Kun Noh da publicação *Attribute-Based Access Control* [54], o controlo de acesso baseado em atributos é uma abordagem flexível que

pode implementar políticas de controlo de acesso, limitadas apenas pela linguagem de programação e riqueza de atributos disponível, tornando-a ideal para sistemas distribuídos ou ambientes que mudam rapidamente.

Esta é uma metodologia de acesso lógica, na qual um utilizador pode realizar uma determinada ação se tiver os atributos necessários para tal [53]. Os atributos podem ser sobre qualquer pessoa ou cenário e costumam enquadrar-se em quatro categorias [3]:

- **Atributos de assunto:** caracterizam o utilizador que está a tentar aceder (os assuntos podem ser idade, autorização, departamento, função, cargo, entre outros);
- **Atributos de ação:** identificam o tipo de ação que o utilizador quer realizar (por exemplo, ler, excluir, visualizar, aprovar...);
- **Atributos do objeto:** descrevem o objeto (ou recurso) a ser acedido, ou o tipo de objeto (tal como ficha médica, conta bancária, departamento, classificação ou sensibilidade, localização...);
- **Atributos contextuais (ambiente):** lidam com o tempo, localização ou aspectos dinâmicos do cenário de controlo de acesso (por exemplo, se um utilizador se encontra em Portugal).

O desenho do sistema pode ser semelhante ao anterior, apenas com Atributos, ao invés de Funções.

Figura 2.4: Modelo de Acesso Baseado em Atributos

Esta solução não resolve os problemas apontados na sub-secção anterior, sendo que é igualmente necessário que as equipas de Front-End tenham conhecimento de todas as permissões e que os gestores percebam que permissões contêm os atributos. A distinção entre o modelo baseado em atributos e o modelo baseado em funções, ficará claro na próxima sub-secção.

2.1.4 Comparação entre modelos

Apesar dos modelos terem a mesma finalidade, distinguem-se no seu desenvolvimento.

Entre as listas de controlo de acesso e o controlo de acesso baseado em funções, as diferenças são evidentes e a sua utilização deve depender inteiramente da complexidade dos requisitos da aplicação. A primeira tem uma implementação mais simples, pois é removida a camada intermédia das funções, sendo feita uma associação direta. Não obstante, as funções ajudam na resolução da adversidade referida na sub-secção 2.1.1, isto é, caso exista uma quantidade considerável de regras de negócio e utilizadores com cargos muito distintos, uma agregação de funções pode ser uma vantagem.

Já o modelo de controlo de acesso baseado em atributos difere do modelo de controlo de acesso baseado em funções no que toca ao conceito de políticas de acesso, na medida em que as funções representam apenas os cargos/papéis na aplicação ou sistema, enquanto que os atributos são mais abrangentes, podendo ter valores complexos definidos, ou valores atómicos, em diferentes categorias [3]. Os atributos são uma extensão do modelo de funções, no entanto, podem ser exatamente iguais se se considerar que um atributo corresponde a uma função.

No FSIM consta o modelo de acesso baseado em funções, dado que a ligação entre utilizadores e funções se adequa precisamente ao efeito pretendido e aos requisitos necessários. Optou-se por uma implementação híbrida do modelo apresentado na secção 2.1.2, usando a Base de Dados, para a associação entre utilizadores e funções, e um ficheiro de configuração YAML, para a ligação entre funções e permissões. Este desenho será explicado no capítulo 4. As vantagens desta implementação híbrida passam pela manutenção da segurança e da fiabilidade das funções dos utilizadores (pois estas estão guardadas em Base de Dados) e pela obtenção de melhor performance na verificação das permissões, que estão sempre carregadas em memória, dado que esse carregamento se dá no arranque da aplicação.

2.2 Bases de Dados

As Bases de Dados são também uma componente de grande importância em desenvolvimento de software, sendo responsáveis pelo armazenamento da informação inserida pelos utilizadores na aplicação.

Surgiram em 1970, quando Edgar F. Codd criou o primeiro modelo relacional [44].

Atualmente existem três tipos de Bases de Dados com características e objetivos distintos: Bases de Dados Relacionais, Bases de Dados Não Relacionais (NoSQL) e Bases de Dados NewSQL (um modelo mais recente) [48].

As Bases de Dados relacionais usam tabelas que contêm colunas e linhas, sendo que as colunas representam os campos que são necessários guardar e cada linha representa uma nova entrada. As tabelas podem ligar-se entre si através de chaves estrangeiras ou colunas comuns [50]. Estas Bases de Dados são baseadas nas propriedades *ACID* (Atomicidade, Consistência, Isolamento, Durabilidade) [50], que permitem obter um elevado grau de fiabilidade, pelo que são uma melhor opção quando a integridade e consistência dos dados é um dos requisitos de um sistema. Todavia, têm um problema de escalabilidade consoante o crescimento dos dados [48].

Como solução, surgiram as Bases de Dados NoSQL que, em desvantagem, não apresentam as propriedades *ACID*, mas que, em seu benefício, contam com as propriedades *BASE* (Basically Available, Soft state, Eventual consistency), tornando-as mais escaláveis e permitindo uma melhor gestão quando existem quantidades massivas de dados.

Estas são divididas em 5 tipos distintos, em conformidade com o seu funcionamento [59]:

- ***Key-Value Store Databases***

Os dados guardados estão associados a uma chave (idêntico ao funcionamento das *Hash-Tables*).

- ***Column-Oriented Databases***

Têm colunas extensíveis que contêm dados relacionados, contrariamente às colunas de Bases de Dados relacionais que são muito estruturadas.

- ***Document Store Databases***

Guardam a informação como coleções de documentos [56].

- ***Graph Databases***

Contêm grafos (conjunto de nós e vértices, em que os nós são os objetos e os vértices as suas relações [59]) na estrutura de dados e a manipulação é também orientada para operações sobre grafos [41].

- ***Object Oriented Databases***

Os dados são tratados como objetos (semelhante ao conceito de objetos na programação orientada a objetos [59]).

O mais recente modelo denominado NewSQL foi, segundo Fatima Haleemunnisa e Kumud Wasnik, desenhado para beneficiar das propriedades das Bases de Dados relacionais, incorporando simultaneamente a verticalidade oferecida pelas Bases de Dados não relacionais [48]. No artigo por eles publicado [48], no qual é demonstrada uma comparação entre os três modelos de Bases de Dados, aplicada à Internet das Coisas, o NewSQL destaca-se como 'vencedor', apesar de existirem algumas ressalvas relativas ao modelo e testes pouco significativos em aplicações empresariais.

Devido à necessidade de consistência de dados no projeto no qual estive a trabalhar, o modelo eleito foi o *PostgresSQL*, uma Base de Dados relacional *opensource* que começou a ser desenvolvida em 1977 [47]. Aliado ao préstimo da consistência de dados, esta oferece a possibilidade de usufruir de uma extensão *opensource* de grande utilidade para aplicação, chamada *Postgis*, que adiciona suporte a *queries* de localização geográfica [28]. No FSIM, não foi necessária a recolha de uma quantidade enorme de informação, pelo que a escalabilidade não constituiu um problema para o projeto.

2.3 Java Spring

No projeto do FSIM foi utilizado Java Spring como *framework open source* para o desenvolvimento do Back-End e do Servidor de Relatórios. Esta representa uma parte muito importante, dado que acelera o desenvolvimento do software, ao fornecer um suporte abrangente à infraestrutura para o desenvolvimento de aplicações Java [55].

Conforme retratado no diagrama 2.5, o Spring compõe vários recursos organizados nos seguintes módulos e camadas [55]:

- *Data Access/Integration* – Camada de acesso a dados.
- *Web* – Camada que fornece ferramentas de integração orientadas para *web*.
- *AOP (Aspect Oriented Programming), Aspects & Instrumentation* – Camadas de programação orientada a aspetos e de instrumentação que dá suporte para o arranque das classes, a ser utilizado em alguns servidores de aplicações.

- *Core Container* – Camada com os módulos centrais do Spring, para a criação da estrutura.

- *Test* – Módulo para execução de testes unitários.

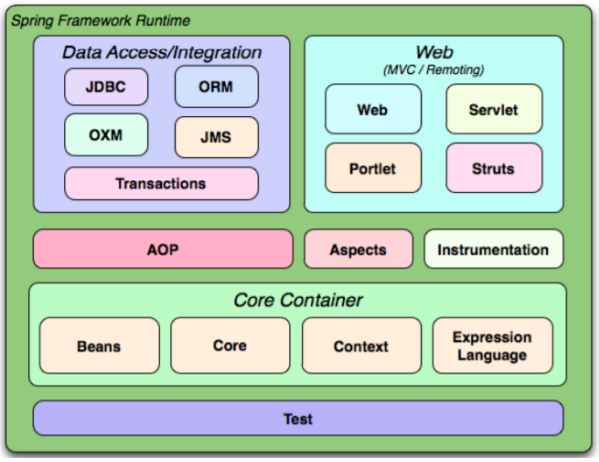

Figura 2.5: Módulos do Spring [55]

Das camadas mencionadas, destaco as utilizadas nas aplicações deste projeto:

Spring Core Container:

Módulo basilar do Spring, necessário para o seu funcionamento e configurado consoante a necessidade. Os módulos Core e Beans fornecem as partes fundamentais da estrutura, incluindo inversão de controlo (IoC) e recursos de injeção de dependência. O Spring transforma as classes com anotações *@Service* em Enterprise Java Beans.

Acesso a Dados:

Oferece suporte a todas as *frameworks* populares de acesso a Base de Dados usadas em Java, tais como *JDBC*, *iBatis/MyBatis*, *Hibernate*, *Java Data Objects (JDO)*, *Java Persistence API (JPA)*, *Oracle TopLink*, *Apache OJB* e *Apache Cayenne*. Permite também a gestão de transações no acesso aos dados, que podem ser configuradas através de anotações, e o preenchimento de um formulário do Spring que origina uma configuração em XML.

Testes:

Permite realizar testes unitários à lógica do Back-End, com *JUnit* ou *TestNG*. Este módulo é essencial ao desenvolvimento de aplicações, dado que, sendo a integração continua, possibilita despistar à partida erros criados por novas implementações.

No módulo Core, o Spring fornece um *container* de inversão de controlo, indispensável para o funcionamento da aplicação FSIM, pois é responsável pelo ciclo de vida dos objetos, desde a sua criação até ao final. Este cria Enterprise Java Beans (EJB) que podem ser configurados de forma a satisfazerem as necessidades da aplicação. Os EJB funcionam como *containers* para encapsular a lógica de negócio, contida nos serviços.

Outro recurso de relevo consiste na injeção de dependências entre módulos, que permite a utilização de outros serviços da aplicação com baixo nível de acoplamento [67]. Deste modo, para utilizar os serviços, não é necessária a criação dos objetos, uma vez que são injetados pelo construtor de forma automática.

2.4 Spring Security vs. Desenvolvimento Interno

Existe, igualmente, uma *framework* que poderia ser integrada no projeto FSIM, denominada *Spring Security*. Trata-se de uma biblioteca de autenticação e autorização poderosa e altamente costumizável, que permite a sua integração de forma simples nas aplicações por parte dos programadores [37]. Dos recursos que fornece, sublinham-se [37]:

- Suporte abrangente e extensível para autenticação e autorização;
- Proteção de ataques como *session fixation*, *clickjacking* e *cross site request forgery*;
- Integração com API;
- Integração Opcional com Spring MVC.

Esta *framework* proporciona um suporte para controlo de acesso baseado em funções, pelo que poderia ter sido empregue para tratar da autorização e autenticação na aplicação FSIM. No entanto, para o efeito, optou-se por fazer de raiz o desenvolvimento de um módulo complementar de gestão de utilizadores (USM), para obter mais controlo sobre o mesmo.

Existem vantagens e desvantagens tanto na utilização de ferramentas *open source* como nos desenvolvimentos fechados. Jaap-Henk Hoepman e Bart Jacobs explicam, no artigo *Increased Security Through Open Source* [52], como a segurança através do segredo foi a metodologia mais utilizada durante anos, quer por empresas quer por serviços militares.

Por um lado, quando se recorre a um software *open source*, este é testado por imensos utilizadores, tornando-o mais completo e seguro. Por outro lado, como todos conseguem observar o seu código fonte (ou pelo menos o design) facilita a descoberta de erros lógicos

por parte de atacantes [52] e deixa o software mais exposto à exploração de vulnerabilidades.

Desenvolver ferramentas próprias tem, de igual forma, as suas desvantagens, a salientar a falta de intensidade, frequência e variedade dos testes. Contudo, um software próprio, ou de código fechado, impede que os atacantes explorem a aplicação [52]).

No FSIM, procedeu-se a um desenvolvimento interno, configurável, pelo facto de poder ser aplicado a outros projetos Java da empresa, ao invés de ser utilizada uma biblioteca *open source* como o Spring Security caso, o módulo USM foi realizado de modo fornecer um serviço de autenticação e um de autorização baseado em funções, tornando a realização de testes de penetração imperativa para garantir o correto funcionamento do sistema.

2.5 Sessão HTTPS e Bearer Tokens

A comunicação entre os computadores dos clientes e aplicação FSIM é feita através do protocolo HTTPS, uma versão segura do protocolo HTTP, que foi introduzido em 1994 devido ao aumento da preocupação dos utilizadores com a segurança e privacidade [60].

O protocolo HTTP não guarda estado, criando dificuldades na gestão da sessão dos utilizadores, pelo que, para contornar esta questão, foram desenhadas sessões HTTP que utilizam *cookies* para identificar os utilizadores quando fazem um pedido [31]. A sessão tem uma data de início – a data em que o utilizador realiza a operação de autenticação – e uma data em que a sessão expira (pode ser variável, sendo mais curta em aplicações cuja segurança é maior, como por exemplo, sites de bancos).

As *cookies* tornaram-se, portanto, essenciais para gerir sessões e utilizadores. São capazes de guardar *tokens* e informações que podem ser usadas pelo Javascript executado no cliente, ou enviadas para o servidor para validações. De acordo com Vipin Samar [63], uma *cookie* é enviada para o cliente com um parâmetro *Set-Cookie* no Cabeçalho HTTP e pode conter 6 elementos: nome, valor, data em que expira, *URL* ao qual pertence, domínio ao qual pertence e *secure* (referente à obrigatoriedade de ser enviada apenas em conexões seguras [33]).

É aqui que entram os *bearer tokens*, que são *strings* criptográficas, normalmente geradas pelo servidor na resposta ao pedido de autenticação [4]. Devem conter informação suficiente em cada pedido HTTP ou HTTPS para o servidor conseguir autenticar o cliente e prosseguir com o pedido.

Dentro dos *bearer tokens* existentes, destaca-se o *JSON Web Token* (JWT), um standard RFC 7519 desenvolvido para criar *tokens* de sessão. Este fornece bibliotecas de utilização para inúmeras linguagens de programação, incluindo Java. Um JWT pode ser enviado no valor de uma *cookie* e abrange três partes separadas por um ponto final [23]:

- Cabeçalho – contém o algoritmo de *hash* utilizado para o *token*;
- *Payload* – um objeto JSON que pode conter qualquer tipo de informação (por exemplo, sobre o utilizador);
- Assinatura – um *hash* para validação da integridade do *token*.

No projeto FSIM deu-se uso ao JWT para identificar os utilizadores da aplicação, com o algoritmo de encriptação HS256 (HMAC com SHA-256) no *header* e com informação pessoal sobre o utilizador (como as suas funções, nome e email) no *payload*.

2.6 Geração Automática de Relatórios

A geração automática de relatórios apresenta uma elevada importância em alguns softwares aplicacionais. Existem várias ferramentas para o efeito, como por exemplo, o *Power BI Reports Server* ou o *JasperReports*.

O *Power BI Reports Server* é uma ferramenta desenvolvida pela Microsoft, que fornece aos utilizadores relatórios avançados e interativos [29], que podem ser desenhados utilizando o *Power BI Desktop*, no qual é possível ter uma pré-visualização dos resultados.

Este software disponibiliza ainda um servidor para a compilação dos relatórios, para serem depois fornecidos aos clientes em diversas plataformas [29]. Contudo, não é *open source*, estando a sua utilização dependente da compra.

Já o *JasperReports* é uma biblioteca de relatórios *open source*, embutida em Java, que oferece a possibilidade de criação de relatórios de uma forma simples [51], tendo sido a opção eleita para o FSIM, por se apresentar como uma melhor solução para os requisitos necessários.

Surgiu em 2001, quando Teodor Danciu necessitou de procurar formas de geração de relatórios e apenas se deparou com ferramentas caras. Desta frustração resultou a ideia de criar o seu próprio compilador de relatórios [51], o qual se tornou um dos mais conhecidos do mundo. É possível criar relatórios em inúmeros formatos utilizando esta biblioteca,

tais como pdf, docx ou xlsx. Tipicamente, o *workflow* da geração segue de acordo com a figura 2.6:

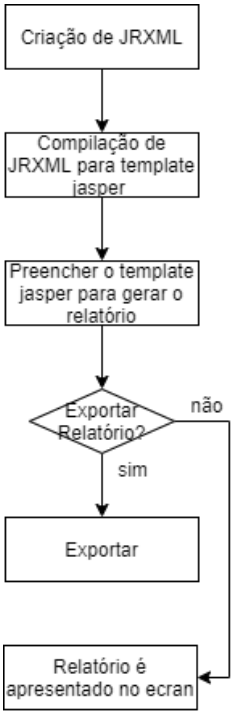

Figura 2.6: Diagrama de *workflow* de geração de relatórios Jasper [51]

O primeiro passo consiste na criação um *template* para o relatório, que será representado num ficheiro JRXML. De seguida, esse *template* é compilado, dando origem a um ficheiro com extensão .jasper que deve ser preenchido com os dados da Base de Dados correta. O documento poderá ser exportado para uma diretoria ou apresentado no ecrã, com a possibilidade de ter diferentes extensões.

Os ficheiros JRXML podem ser criados utilizando o software Jasper Studio, próprio para desenhar e compilar os *templates*, construir *queries* do relatório, entre outras funcionalidades. É possível elaborar documentos com qualquer complexidade de dados e aceder a várias fontes de dados como CSV, XML ou JDBC [19].

O Jasper Studio permite ainda a integração com o JasperReports Server, um servidor de relatórios autónomo e incorporável, que, além de relatórios, fornece dados analíticos. Este pode ser um componente introduzido numa aplicação *web* ou pode funcionar como um *hub* central de armazenamento de informação [18].

Na aplicação FSIM, optou-se por não utilizar o Jasper Server devido aos constrangimentos que poderia trazer no controlo de versões dos relatórios durante o desenvolvimento e no processo de integração contínua da equipa, tendo, a partir desse cenário, surgido a

necessidade de criar um servidor de relatórios escrito em Java. A análise deste tema será feita em detalhe no capítulo 3.

2.7 Testes de Penetração

Outra matéria fundamental em qualquer projeto de Engenharia de Software consiste na garantia da segurança. Uma das metodologias existentes para testar a segurança de software é através de testes de penetração, feitos com o objetivo de descobrir vulnerabilidades e assegurar a proteção da informação [70].

Um dos tópicos mais importantes hoje em dia, dada a exposição do software, é a segurança. O que significa então a *segurança de software*? Segundo Gary McGraw [57], segurança de software é a capacidade que um software tem de continuar a funcionar de forma correta, sob ataques maliciosos. O autor refere ainda que a segurança deve ser desenhada no início do ciclo de vida do software, sendo importante compreender bem as ameaças de segurança [57]. Para conhecê-las melhor atualmente, destaco o trabalho do *Open Web Aplication Secure Project* (OWASP), uma organização sem fins lucrativos focada em melhorar a segurança de software. A sua missão consiste em aumentar a visibilidade da segurança de software e ajudar programadores e organizações a tomar decisões conscientes relativamente à proteção das suas aplicações [26].

A organização tem vários projetos *open source*, incluindo o OWASP Top 10 [69] que enumera as 10 vulnerabilidades mais críticas:

- **Injeção** – Falhas de injeção (como injeções SQL, NoSQL, OS e LDAP) que ocorrem quando dados não confiáveis são enviados como parte de um comando ou *query*;
- **Quebra de Autenticação** – Possibilidade de um atacante comprometer passwords, chaves ou *tokens* de sessão e explorar outras falhas de implementação para assumir as identidades de utilizadores, temporária ou permanentemente;
- **Exposição de Dados Sensíveis** – Um atacante pode roubar ou modificar dados sensíveis fracamente protegidos para cometer fraudes de cartão de crédito, roubo de identidade ou outros crimes;
- **Entidades Externas de XML (XXE)** – Ataque realizado a processadores de XML, tornando possível aos atacantes a alteração ou upload de ficheiros XML criados por eles;

- **Quebra de Controlo de Acessos** – Exploração de falhas nas permissões dos utilizadores para aceder e modificar funcionalidades e/ou dados não autorizados;

- **Configurações de Segurança Incorretas** – Configurações erradas ou configurações por defeito permitem que sejam exploradas vulnerabilidades;

- ***Cross-Site Scripting (XSS)*** – Atacantes conseguem executar *scripts* no browser de clientes, podendo roubar sessões de utilizador, desfigurar websites ou redirecionar o utilizador para sites maliciosos;

- **Desserialização Insegura** – Ocorre quando um atacante consegue colocar dados não estruturados na aplicação, podendo levar à execução e injeção remota de código ou à negação de serviço;

- **Utilização de Componentes com Vulnerabilidades Conhecidas** – Se um componente vulnerável (como bibliotecas, *frameworks* e outros módulos de software) for explorado, o ataque pode facilitar a perda de dados graves ou o controle do servidor;

- **Registo e Monitorização Insuficiente** – Falta de monitorização e uma resposta ineficaz a incidentes permitem que os invasores ataquem ainda mais os sistemas e adulterem, extraiam ou destruam dados.

Os testes de penetração, normalmente realizados por engenheiros de segurança com a devida autorização [70], devem, portanto, ser desenhados de forma a tentar explorar estas possíveis vulnerabilidades nas aplicações.

É comum serem usadas ferramentas de software por parte de quem realiza testes de penetração, nomeadamente as que são utilizadas por *hackers* maliciosos [70] (como o OWASP ZAP ou SFUZZ, em detalhe abaixo). Brad Arkin, et al [42], salienta o uso destes recursos como uma boa prática, sobretudo por dois motivos: o primeiro prende-se com o facto de realizarem grande parte do trabalho de testes de forma mais autónoma, quando usadas eficazmente; o segundo passa pela facilidade de transformação dos *outputs* em métricas, o que torna mais simples a sua análise pelas equipas de desenvolvimento. Os autores apontam também aspetos negativos, relacionados com o facto de não existirem instrumentos que verifiquem vulnerabilidades ao nível do design, tendo esse trabalho de ser realizado por especialistas.

Existem dois tipos distintos de ferramentas: ferramentas de análise estática e ferramentas de análise dinâmica.

2.7.1 Ferramentas de Análise Estática

A análise estática de software ocorre quando a ferramenta tem acesso ao código e procura pela utilização de bibliotecas perigosas e vulnerabilidades conhecidas no código.

Podemos encontrar uma panóplia de recursos que analisam vulnerabilidades de software estaticamente. A título de exemplo, pode destacar-se o *Sonarqube*, o *Kiuwan* e o *EvoMaster*.

O *Sonarqube* é um software capaz de analisar código e detetar erros, vulnerabilidades de segurança, pontos de entrada e ainda *code smells* [8]. Desenvolvido pela Sonarsource, trata-se de uma ferramenta de *open source*, apta para realizar análise estática de projetos escritos em mais de 20 linguagens de programação.

O *Kiuwan* aplica mais de 4000 regras baseadas nos standards da indústria de segurança, incluindo OWASP Top 10 e CWE/SANS [39].

O *EvoMaster* cria testes unitários recorrendo à definição *swagger* da API, fornece deteção de falhas e heurísticas avançadas de caixa branca (com acesso ao código) [14]. Esta pode ser, de igual forma, considerada uma ferramenta de análise dinâmica, uma vez que também oferece testes de caixa preta (sem acesso ao código) com a aplicação em execução.

2.7.2 Ferramentas de Análise Dinâmica

A análise dinâmica, também conhecida como *fuzzing*, é uma técnica de realização de testes de segurança de caixa preta com a aplicação em execução, não sendo necessário ter o código fonte. O objetivo passa por encontrar vulnerabilidades através da injeção de código malicioso ou mal-formado de modo automático [13]. Assim, *fuzzing* é o processo de enviar dados inválidos de forma intencional para uma aplicação ou sistema, no sentido de provocar um erro ou falta [68].

OWASP-ZAP é um exemplo de uma ferramenta gratuita de segurança de software utilizada para o efeito e a sua manutenção é assegurada pela comunidade internacional. Serve o propósito de encontrar vulnerabilidades de uma aplicação automaticamente, na fase de desenvolvimento, sendo também uma ótima ferramenta para engenheiros de segurança que realizam testes de penetração [27].

Outro instrumento existente é o SFUZZ (*Simple Fuzz*) que cria cenários de testes de penetração, sem ser necessário grandes conhecimentos de programação [35]. Este dis-

ponibiliza vários serviços, como a criação de *scripts* para cenários de testes, inclusão de conteúdo de pacotes anteriores e *payloads* de transporte para UDP e TCP.

2.8 XML vs. JSON vs. YAML

O XML (Extensible Markup Language) é um formato de texto simples e flexível, derivado do SGML (ISO 8879) [11]. Podemos observar um exemplo de configuração em XML abaixo:

```
1       <carta>
2           <para>Paulo</para>
3           <de>Francisca</de>
4           <titulo>Carta de Exemplo</titulo>
5           <corpo>Carta configurada em XML</corpo>
6       </carta>
```

Este formato é usado em serviços *web* SOAP. No entanto, estes têm vindo a ser substituídos por REST que utiliza JSON como formato de resposta.

O JSON (Java Script Object Notation) é uma linguagem leve, de simples leitura/escrita por humanos e fácil de ser lida/gerada por máquinas [21]. Segue-se o mesmo exemplo de configuração acima demonstrado, desta vez em JSON:

```
1 {
2   "carta": [
3     {
4       "para": "Paulo",
5       "de": "Francisca",
6       "titulo": "Carta de Exemplo",
7       "corpo": "Carta configurada em JSON"
8     }
9   ]
10 }
```

O YAML (*YAML Ain't Markup Language*) é uma ferramenta de serialização de dados semelhante ao JSON, no que diz respeito à fácil compreensão e escrita por humanos [25], bem como à sua estrutura. Contudo, utiliza o caracter '–' para representação de listas, ao invés de chavetas ou parênteses.

A mesma configuração em YAML pode ser representada da seguinte forma:

```
1  - carta:
2      para: Paulo
3      de: Francisca
4      titulo: Carta de Exemplo
5      corpo: Carta configurada em YAML
```

XML, JSON e YAML são três formatos de ficheiros distintos, com objetivos idênticos. Estes são usados muitas vezes para configurações de um sistema, dado a existência de bibliotecas para a sua transformação em objetos da linguagem de programação respetiva.

Capítulo 3

Análise

Este capítulo encontra-se omisso por motivos de confidencialidade.

Capítulo 4

Desenho

Este capítulo encontra-se omisso por motivos de confidencialidade.

Capítulo 5

Implementação

Este capítulo encontra-se omisso por motivos de confidencialidade.

Capítulo 6

Testes e Resultados

Este capítulo encontra-se omisso por motivos de confidencialidade.

Capítulo 7

Conclusão

O projeto decorreu conforme planeado, alcançando todos os objetivos propostos com sucesso. O FSIM, criado como uma ferramenta importante de apoio à certificação ambiental, já se encontra em produção. A aplicação foi desenvolvida de forma robusta, segura e completa, permitindo a realização de múltiplas tarefas de gestão, através de todos os serviços implementados ao longo do projeto.

Enquanto parte integrante da equipa, contribuí ativamente para o desenvolvimento do módulo USM, responsável por gerir os utilizadores, nomeadamente no que toca a gestão de permissões, recorrendo a um modelo de funções e permissões. Concisamente, o modelo proposto está definido por camadas – utilizadores, funções e permissões – e permite a atribuição de funções aos diferentes utilizadores da aplicação que, por sua vez, contêm permissões para realizar determinadas ações na plataforma. Deste modo, foi possível satisfazer os requisitos funcionais, de forma simples e eficaz.

Produzi, igualmente, um servidor de relatórios Jasper escrito em Java Spring, cujas condições são explicadas no capítulo 3. Esta aplicação complementar está, atualmente, a ser utilizada noutros projetos da empresa, uma vez que a sua configuração e instalação é simples e apresenta várias vantagens, a salientar, a possibilidade de ser incluída em todo o ciclo de vida do desenvolvimento do software com integração contínua. A aplicação permite a geração automática de relatórios, utilizando *templates* Jasper.

Parte das minhas contribuições englobaram também a criação de um serviço de importação de membros, capaz de ler um ficheiro colocado num *S3 bucket* no AWS e de criar novos membros com a informação lida. Este foi, de igual forma, um componente de extrema importância para o FSIM, dado que a inserção manual de membros por parte de todos os clientes seria um processo demasiado moroso e custoso.

Durante este projeto de Engenharia Informática, propus-me ainda à realização de testes

de penetração, para garantir a robustez da aplicação, essencialmente do módulo USM. Durante estes testes, não foram encontradas vulnerabilidades de risco elevado com necessidade de correção imediata, com exceção de algumas falhas pouco significativas, abordadas na secção 7.2 infra. As metodologias utilizadas, bem como os resultados, podem ser observados em mais detalhe no capítulo 6.

Desenvolvi ainda outros serviços para completar a API, como a importação de membros, o PGOA e a geração de temas para visualização. Para estes desenhei, de igual forma, testes unitários para serem executados no processo de integração contínua.

Os desenvolvimentos mencionados foram também testados a nível funcional e aceites pela equipa de testes.

As tarefas compreendidas no plano inicial de trabalhos foram cumpridas e bem sucedidas, tendo sido destacado como *team-leader* numa fase final do projeto, para o acompanhamento após entrada em produção. As minhas responsabilidades passavam por identificar as questões criadas pelos clientes no Jira, atribuir tarefas à equipa de FE, resolver tarefas de BE, coordenar as reuniões diárias da equipa de desenvolvimento e, no rescaldo do estágio, realizar testes funcionais sobre todo o trabalho efetuado pela equipa.

7.1 Dificuldades Encontradas

De uma forma geral não existiram grandes constrangimentos ao longo do desenvolvimento dos serviços e funcionalidades da aplicação.

Além das contribuições especificadas acima, transcrevi para Java um sistema capaz de criar um ficheiro de configuração de temas para visualização. O processo de transcrição do código de Python para Java revelou-se a maior dificuldade encontrada. No entanto, o serviço foi criado e está a ser utilizado atualmente, sendo uma mais-valia para a arquitetura da aplicação por não necessitar de um servidor extra de Python ou NodeJs.

Outro inconveniente surgiu durante o desenvolvimento da aplicação de relatórios. Pelo facto de ser uma aplicação externa ao BE, durante a realização de testes no ambiente de Desenvolvimento, a falta de informação nos *logs* não possibilitava perceber o motivo pelo qual, por vezes, os relatórios não eram gerados. A solução passou por aumentar a informação presente nos *logs* e, face à necessidade de arranjar um mecanismo para confluir os *logs* do BE com os do servidor de relatórios, enviar um *uuid* como parâmetro de *query string* no pedido do BE. Atualizou-se o controlador Java para passar a ter este parâmetro obrigatório e, como tal, no início do pedido na aplicação é impresso um *log* que

contém esse identificador único. Através dessa informação, percebemos nos *logs* que os relatórios em pdf não eram gerados, pois a versão do Java Runtime Environmnent (JRE) escolhido para o *docker* não tinha todas as bibliotecas necessárias. A correção foi apenas colocar uma versão do JRE mais completa.

Ainda no que diz respeito ao servidor de relatórios, manifestaram-se algumas adversidades relativas às bibliotecas Maven utilizadas para a geração de relatórios pdf. Na fase inicial, encontrei uma biblioteca que continha os métodos essenciais para a geração de pdfs, no entanto, os mesmos eram devolvidos em branco. Após alguma pesquisa, descobri que os serviços necessários para o efeito apenas funcionavam numa versão mais antiga do repositório Maven que estava a tentar aceder.

7.2 Trabalho Futuro

Os futuros trabalhos serão no sentido de tornar a aplicação ainda mais segura e robusta.

O capítulo 6 evidenciou que o nosso *token* poderia ser quebrado com maior poder computacional. As soluções identificadas foram:

- utilizar outro algoritmo para o *token* (HS512, ao invés do HS256)
- adicionar o campo com os códigos das funções do utilizador ao campo que não se encontra em texto claro

Estas alterações não foram implementadas, encontrando-se ainda em estudo. Está a ser discutida a atualização do algoritmo para tornar o *token* mais protegido. Para tal, será necessário atualizar também a biblioteca JWT utilizada de forma a conter o algoritmo, alterar a verificação da assinatura e, posteriormente, elaborar testes de performance para perceber o impacto causado pela mudança, uma vez que o *token* é utilizado em todos os pedidos ao BE da aplicação. A segunda proposta de resolução está igualmente em consideração, pois é necessário analisar as repercussões de uma reversão ao *hash* presente no *token* em todos os pedidos e perceber se fará sentido aplicá-la em simultâneo com a outra solução mencionada. Em todo o caso, ambas serão possivelmente testadas em projetos futuros, antes de serem postas em prática no FSIM .

Outra tarefa que deverá ser realizada futuramente consiste numa alteração de configuração sobre os protocolos permitidos pela aplicação. Após testes realizados com recurso ao site *www.ssllabs.com* [38], observou-se que o FSIM obteve nota B. Analisando os dados, concluiu-se que o resultado derivou da utilização de alguns protocolos antigos de TLS 1.1

e 1.2. Através de modificações na configuração do AWS, foi-nos possível melhorar a nota para A num dos ambientes de testes. Todavia, é necessário perceber junto dos clientes se terá algum efeito negativo, caso estes não utilizem *browsers web* atualizados.

Bibliografia

[1] Amazon ec2 t3 instances – amazon web services (aws). https://aws.amazon.com/ec2/instance-types/t3/.

[2] Amazon rds instance types — cloud relational database — amazon web services. https://aws.amazon.com/rds/instance-types/.

[3] Attribute-based access control - wikipedia. https://en.wikipedia.org/wiki/Attribute-based_access_control.

[4] Bearer authentication. https://swagger.io/docs/specification/authentication/bearer-authentication/.

[5] Brute forcing hs256 is possible: The importance of using strong keys in signing jwts. https://auth0.com/blog/brute-forcing-hs256-is-possible-the-importance-of-using-strong-keys-to-sign-jwts/.

[6] Cloud object storage — store & retrieve data anywhere — amazon simple storage service (s3). https://aws.amazon.com/s3/.

[7] Crudrepository (spring data core 2.3.3.release api). https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html.

[8] Documentation — sonarqube docs. https://docs.sonarqube.org/latest/.

[9] Easyjet says cyberattack stole data of 9 million customers - the new york times. https://www.nytimes.com/2020/05/19/business/easyjet-hacked.html.

[10] Expresso — covid-19 faz disparar ataques de hackers em portugal. https://expresso.pt/coronavirus/2020-05-14-Covid-19-faz-disparar-ataques-de-hackers-em-Portugal.

[11] Extensible markup language (xml). https://www.w3.org/XML/.

[12] Fsc produces positive environmental impacts – university research — forest stewardship council. https://fsc.org/en/newsfeed/fsc-produces-positive-environmental-impacts-university-research.

[13] Fuzzing — owasp. https://owasp.org/www-community/Fuzzing.

[14] Github - emresearch/evomaster: A tool for automatically generating system-level test cases. currently targeting rest apis. https://github.com/EMResearch/EvoMaster.

[15] Inoweiser - our ecosystem. https://www.inoweiser.com/our-ecosystem.

[16] Introduction to spring mvc handlerinterceptor — baeldung. https://www.baeldung.com/spring-mvc-handlerinterceptor.

[17] The ioc container. https://docs.spring.io/spring-framework/docs/3.0.x/spring-framework-reference/html/beans.html#beans-factory-scopes.

[18] Jasperreports® server — jaspersoft community. https://community.jaspersoft.com/project/jasperreports-server.

[19] Jaspersoft® studio — jaspersoft community. https://community.jaspersoft.com/project/jaspersoft-studio.

[20] Jenkins user documentation. https://www.jenkins.io/doc/.

[21] Json. https://www.json.org/json-en.html.

[22] Json web tokens - jwt.io. https://jwt.io/.

[23] Learn how json web tokens (jwt) works in just a few minutes — by dler ari — medium. https://medium.com/@dleroari/learn-the-basics-of-json-web-tokens-jwt-and-how-it-works-in-practice-8b3b14cbe0f9.

[24] Mime types - http — mdn. https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Basico_sobre_HTTP/MIME_types.

[25] The official yaml web site. https://yaml.org/.

[26] Owasp foundation, the open source foundation for application security. https://owasp.org/.

[27] Owasp zap zed attack proxy — owasp. https://owasp.org/www-project-zap/.

[28] Postgis — spatial and geographic objects for postgresql. https://postgis.net/.

[29] Power bi report server — microsoft power bi. https://powerbi.microsoft.com/pt-pt/report-server/.

[30] qwc2-demo-app/readme.md at master · qgis/qwc2-demo-app · github. https://github.com/qgis/qwc2-demo-app/blob/master/README.md.

[31] Rfc 2109 - http state management mechanism. https://tools.ietf.org/html/rfc2109.

[32] Role-based access control - wikipedia. https://en.wikipedia.org/wiki/Role-based_access_control#cite_note-6.

[33] Secure cookie flag control — owasp foundation. https://owasp.org/www-community/controls/SecureFlag.

[34] Semantic versioning 2.0.0 — semantic versioning. https://semver.org/.

[35] sfuzz — penetration testing tools. https://tools.kali.org/vulnerability-analysis/sfuzz.

[36] Spring initializr. https://start.spring.io/.

[37] Spring security. https://spring.io/projects/spring-security.

[38] Ssl server test (powered by qualys ssl labs). https://www.ssllabs.com/ssltest/.

[39] Static code analysis - kiuwan. https://www.kiuwan.com/static-code-analysis/.

[40] Trustsystems. https://www.trustsystems.pt/empresa/.

[41] Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Computing Surveys (CSUR)*, 40(1):1–39, 2008.

[42] Brad Arkin, Scott Stender, and Gary McGraw. Software penetration testing. *IEEE Security & Privacy*, 3(1):84–87, 2005.

[43] David Bernstein. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84, 2014.

[44] E. F. T. Codd. A relational model of data for large shared data banks, Junho 1970.

[45] Fábio Cruz. *Scrum e PMBOK unidos no Gerenciamento de Projetos*. Brasport, 2013.

[46] Erica Di Girolami and BJM Arts. Environmental impacts of forest certifications. Technical report, Forest and Nature Conservation , WU, 2018.

[47] Joshua D Drake and John C Worsley. *Practical PostgreSQL*. "O'Reilly Media, Inc.", 2002.

[48] Haleemunnisa Fatima and Kumud Wasnik. Comparison of sql, nosql and newsql databases for internet of things. In *2016 IEEE Bombay Section Symposium (IBSS)*, pages 1–6. IEEE, 2016.

[49] David Ferraiolo, D Richard Kuhn, and Ramaswamy Chandramouli. *Role-based access control*. Artech House, 2003.

[50] Christoforos Hadjigeorgiou et al. Rdbms vs nosql: Performance and scaling comparison. *MSc in High*, 2013.

[51] David R Heffelfinger. *JasperReports 3.5 for Java Developers*. Packt Publishing Ltd, 2009.

[52] Jaap-Henk Hoepman and Bart Jacobs. Increased security through open source. *Communications of the ACM*, 50(1):79–83, 2007.

[53] Vincent C Hu, David Ferraiolo, Rick Kuhn, Arthur R Friedman, Alan J Lang, Margaret M Cogdell, Adam Schnitzer, Kenneth Sandlin, Robert Miller, Karen Scarfone, et al. Guide to attribute based access control (abac) definition and considerations (draft). *NIST special publication*, 800(162), 2013.

[54] Vincent C Hu, D Richard Kuhn, David F Ferraiolo, and Jeffrey Voas. Attribute-based access control. *Computer*, 48(2):85–88, 2015.

[55] Rod Johnson, Juergen Hoeller, Keith Donald, Colin Sampaleanu, Rob Harrop, Thomas Risberg, Alef Arendsen, Darren Davison, Dmitriy Kopylenko, Mark Pollack, et al. The spring framework–reference documentation. *interface*, 21:27, 2004.

[56] Neal Leavitt. Will nosql databases live up to their promise? *Computer*, 43(2):12–14, 2010.

[57] Gary McGraw. Software security. *IEEE Security & Privacy*, 2(2):80–83, 2004.

[58] Anthony Scott Moran, Brian James Turner, and Peter Sean Calvert. Grouped access control list actions, May 27 2008. US Patent 7,380,271.

[59] Ameya Nayak, Anil Poriya, and Dikshay Poojary. Type of nosql databases and its comparison with relational databases. *International Journal of Applied Information Systems*, 5(4):16–19, 2013.

[60] David Naylor, Alessandro Finamore, Ilias Leontiadis, Yan Grunenberger, Marco Mellia, Maurizio Munafò, Konstantina Papagiannaki, and Peter Steenkiste. The cost of the"s"in https. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 133–140, 2014.

[61] Sylvia Osborn, Ravi Sandhu, and Qamar Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(2):85–106, 2000.

[62] Nicholas Pippenger. The shortest disjunctive normal form of a random boolean function. *Random Structures & Algorithms*, 22(2):161–186, 2003.

[63] Vipin Samar. Single sign-on using cookies for web applications. In *Proceedings. IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'99)*, pages 158–163. IEEE, 1999.

[64] Pierangela Samarati and Sabrina Capitani de Vimercati. Access control: Policies, models, and mechanisms. In *International School on Foundations of Security Analysis and Design*, pages 137–196. Springer, 2000.

[65] Ravi S Sandhu and Pierangela Samarati. Access control: principle and practice. *IEEE communications magazine*, 32(9):40–48, 1994.

[66] Marcel Schoffelmeer. Implementing role based security in a web app — bluecore engineering. `https://medium.com/bluecore-engineering/implementing-role-based-security-in-a-web-app-89b66d1410e4`, 2019.

[67] Mark Seemann. *Dependency injection in. NET*.

[68] Michael Sutton, Adam Greene, and Pedram Amini. *Fuzzing: brute force vulnerability discovery*. Pearson Education, 2007.

[69] Dave Wichers and Jeff Williams. Owasp top-10 2017. *OWASP Foundation*, 2017.

[70] John Yeo. Using penetration testing to enhance your company's security. *Computer Fraud & Security*, 2013(4):17–20, 2013.

Apêndice A

Tabela de Permissões Aplicação APP2

Este apêndice encontra-se omisso por motivos de confidencialidade.

Apêndice B

Tabela de Permissões Aplicação APP1

Este apêndice encontra-se omisso por motivos de confidencialidade.

Apêndice C

Tabela de campos de importação de membros

Este apêndice encontra-se omisso por motivos de confidencialidade.